Final Exam Planner

Jan Lishak – 294322
Lenka Orincakova - 293085
Juan Iglesias Trebolle – 293143
João Bernardo Baptista Vieira Dias – 293133

Supervisors: Astrid Hanghøj, Michael Viuff

Number of characters: 3559

Software Technology Engineering

1st semester

16.12.2019

Table of content

Abstract

The Exam Planner system is a tool that manages all the data for creating the exam periods for VIA University College. This system is an improvement in efficiency, since it makes the task of creating exams easier and intuitive. The problem of creating exam schedules is caused by humans, due to the amount of data that they need to handle and all the details that need to be taken into account. It is important to solve this problem because it is wasting time that would be better invested doing other things.

The project was done following a flexible Waterfall method, which is divided in 4 phases: Analysis, where the requirements of the system are stated, Design, with a description of the system based on diagrams, Implementation, with code snippets explained, and Testing, where the requirements are tested within the system.

The application has limitations, it does not contact the service department by itself.
The system will be transformed into a client-server system, to improve the capacity that the users have over the system, from their personal computers.

1 Introduction

The Exam Planner is a system created specifically to meet the requirements that Via University College needs in order to make their exam schedules. Since their current system is not able to avoid human errors and other external factors, this new one will make the job easier and faster, thus improving the administrator’s efficiency.

The new system will detect if each exam is suitable for a specific date, taking into account all the specific criteria. After saving all the data about student, room and examiners, the system will upload the information to the webpage in a table format, so everybody has easy access to this information. On the other side, the system will not contact the service department, instead this will be called by a third party.

All the technical process will be found in this Project Report, starting the Analysis, where the requirements will be set, going through the Design, where it is outlined how the system is structured. Furthermore, interesting code snippets are explained in Implementation, Testing, where the functional requirements are checked in the system, followed by Results and Discussion, based on the testing part, the outcome and achieved results of the project are presented. In Conclusion, the results from each section in the report are compiled and finally, Project Future, where reflections about improvements on the project can be found.

2 Analysis

VIA University College needs to plan exam schedule before the examination starts. The schedule plan must meet many specific criteria. Scheduling in general is simple task but when there are so many conditions for the right plan, the task can become very complex. Computers might possibly help to speed up the process of scheduling.

Our system is able to do: 1) the administrator collects the information, 2) the administrator is uploading the information and system checks the requirements, 3) the schedule is created. The system contains information about students, co-examiners and examiners. The schedule must be flexible and delivered to the students in a way that the information is always updated and easy to access. It would be better if the schedule could be viewed by a student or a teacher in a way that it filters all irrelevant information.

2.1 Requirements

In this section, functional and non-functional requirements, given by the customer, are set.

2.2 Functional Requirements

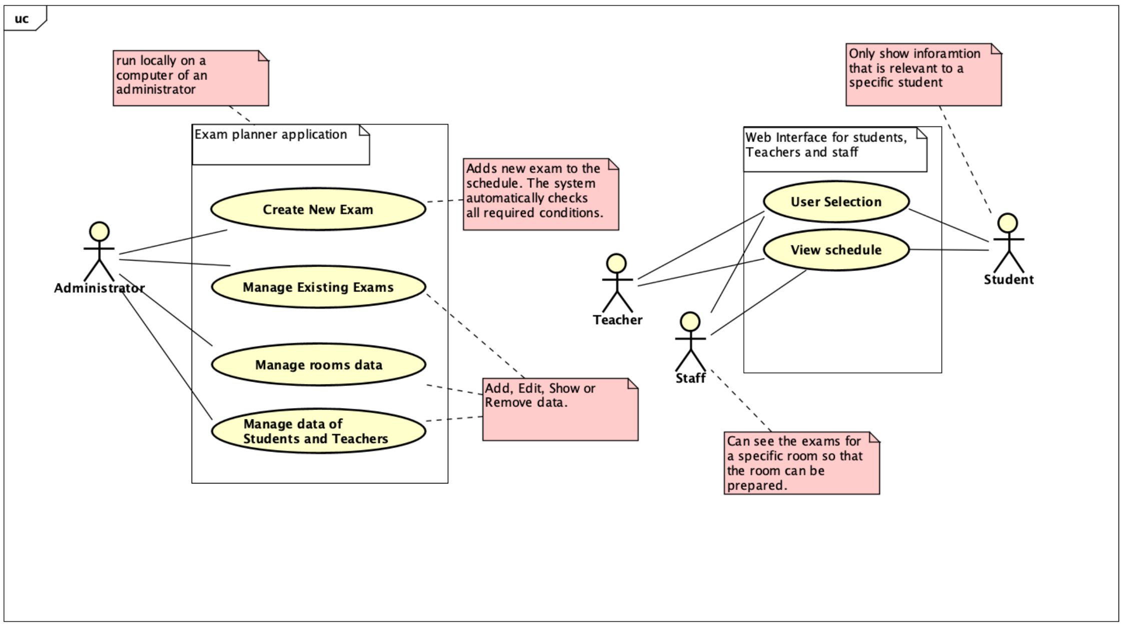
1. As an administrator, I want to be registered as only administrator with access to all information.
2. As an administrator, I want to be able to upload information about teachers, students and classrooms.
3. As an administrator, I want the system to check if there are two exams in the same day, with the right amount of time between them.
4. As an administrator, I want the system to check if the rooms are suitable for exams and if the examiners and co-examiners are available.
5. As an administrator, I want the system to check that the examiners' and co-examiners' workday is 8 hours max.
6. As an administrator, I want the system to check if the written exam is before oral exam.

7. As an administrator, I want the system to check if the time of the written exam is 3 hours and the time of the oral exam is 20 minutes.
8. As an administrator, I want the system to check if the exam on 7th semester is at least 3 days before graduation.
9. As an administrator, I want the system to check if the class for the exam is the same as class used during the semester.
10. As an administrator, I want the system to check if the service department is informed about the exam at least one day in advance to prepare the room for exam.
11. As an administrator, I want to be able to add, remove, edit and show information about exam.
12. As an administrator, I want to be able to add, remove, edit and show information about person.
13. As an administrator, I want to be able to add, remove, edit and show information about room.
14. As a student, I want to be registered as student with access to the schedule.
15. As a student, I want to be able to see the schedule.
16. As a teacher, I want to be registered as a teacher with access to the schedule.
17. As a teacher, I want to be able to see the schedule.

2.3 Non-Functional Requirements

1. Every update in the system includes writing to a file.

2.4 Use case Diagram


uc
run locally on a computer of an administrator
Exam planner application
Create New Exam
Manage Existing Exams
Manage rooms data
Manage data of Students and Teachers
Administrator
Adds new exam to the schedule. The system automatically checks all required conditions.
Add, Edit, Show or Remove data.
Teacher
Staff
Web Interface for students, Teachers and staff
User Selection
View schedule
Student
Only show inforamtion that is relevant to a specific student
Can see the exams for a specific room so that the room can be prepared.

2.5 Use case descriptions

Use case	**Manage room data**
Summary	Add a new classroom, show the list of all classrooms, edit classroom's properties or delete a classroom.
Actor	Administrator.
Precondition	None
Postcondition	A new classroom has been added to the list of classrooms, data for an existing classroom has been shown or updated or an existing classroom has been deleted from the list.
Base sequence	ADD: 1. If a new classroom needs to be added, specify the values for a. Name of the classroom b. The capacity (numbers of places to sit for students) c. Subject taught in this classroom d. Functionalities for projection (HDMI, VGA) 2. System validates data and prompts for illegal values 3. If the values are valid, the system adds a new student with given information. SHOW: 4. Show a list of all classrooms recorded in the system. For each classroom show the defined values as in step number 1. EDIT: 5. The system shows all classrooms as in step number 2. 6. Select which of the classroom properties defined in step number 1 are required to change 7. Update the property with a new valid value REMOVE: 8. Show a list of all classrooms recorded in the system. For each classroom show the defined values as in step number 1. 9. Select which of the existing classroom needs to be deleted. 10. Remove the specified classroom from the list of classrooms.

Use case	**Manage the data of students**
Summary	Add, edit or remove users of the system.
Actor	Administrator
Precondition	None
Postcondition	Information about students is set.
Base sequence	SHOW: 1. show a list of all students recorded in the system. 2. For each student show the first name, last name, ID, subject. ADD: 3. If adding new student, then set up this information: a. ID b. First name c. Last name d. subject 4. System validates data and prompts for illegal values 5. If the values are valid, the system adds a new student with given information. EDIT: 6. System shows a list of all students and all information as defined from step 1. to step 2. 7. Enter or edit values for one of the fields of the student. 8. System updates the selected field of the student. REMOVE: 9. System shows a list of all users and all information as defined from step 1. to step 2. 10. Select a specific student from the list to be deleted 11. System removes the student from the list.

Use case	**Manage the data of teachers.**
Summary	Add, edit or remove users of the system.
Actor	Administrator
Precondition	None
Postcondition	Information about teachers is set.
Base sequence	SHOW: 1. show a list of people recorded in the system. 2. For each teacher show the name, ID, teaching subject, ADD: 3. If adding new teacher, then set up this information: a. ID b. First name c. Last name d. Subjects 4. The teacher must tick a label called “Teacher”. Else, the person will be recognized as student. 5. System validates data and prompts for illegal values 6. If the values are valid, the system adds a new teacher with given information. EDIT: 7. System shows a list of all users and all information as defined from step 1. to step 2. 8. Enter or edit values for one of the fields of the teacher. 9. System updates the selected field of the teacher. REMOVE: 10. System shows a list of all teachers and all information as defined from step 1. to step 2. 11. Select a specific teacher from the list to be deleted 12. System removes the teacher from the list.

Use case	**Manage exam data**
Summary	Add, edit, remove or view exam data.
Actor	Administrator
Precondition	The student's, room's, teacher's and subject data must be already set up.
Postcondition	The data in the schedule is replaced with the one specified.
Base sequence	ADD: 1. Administrator chooses examination period. The list of possible dates will be shown. 2. System shows list of existing subjects and rooms. Administrator chooses one option from these fields. 3. If adding new exam, then set up this information: a. Teacher b. Type of exam (Ordinary/Reexam) c. Format (Written/Oral) d. ECTS e. External examiners 4. If the date is not available for particular room and subject, the list of date will be shown without the date. 5. If the semester is 7th and the date is not three days before graduation, go to step 1 and choose another date. 6. System validates data and prompts for illegal values. 7. If the input is valid then the system adds a new exam with given information. SHOW: 8. System shows a list of exams with all information from step 1. EDIT: Step 1-5 as above and then 9. Enter or edit values for one of the exam fields from step 1. 10. System updates the selected field in the exam. REMOVE: Step 1-5 as above and then 11. Verify deleting the selected exam. 12. System approves the deletion and removes the exam from the list.

Use case	User Selection
Summary	User is asked to enter an ID number to view the schedule.
Actor	Staff, Students and Teachers.
Precondition	Open the webpage.
Postcondition	The user can view all the relevant information about the exams.
Base sequence	1. System asks for type of user (student, teacher or stuff).

Use case	View schedule
Summary	Showing the schedule to students, teachers and stuff.
Actor	Students, teachers, stuff.
Precondition	User must select the type of user.
tcondition	The user can view information about the exams.
Base sequence	1. System checks, what type of user is logged in 2. Accordingly, to the type of user any irrelevant information is filtered out so that the schedule is easily readable. (the data is displayed as a table)

2.6 Link between requirements and use cases

Use case	Covered requirements
Manage room data	2,4,9,13
Manage person data	2, 5, 12
Manage exam data	3,4, 6,7,8,11
User selection	1
View schedule	14,15,16,17

2.7 Activity diagram (Manage rooms)

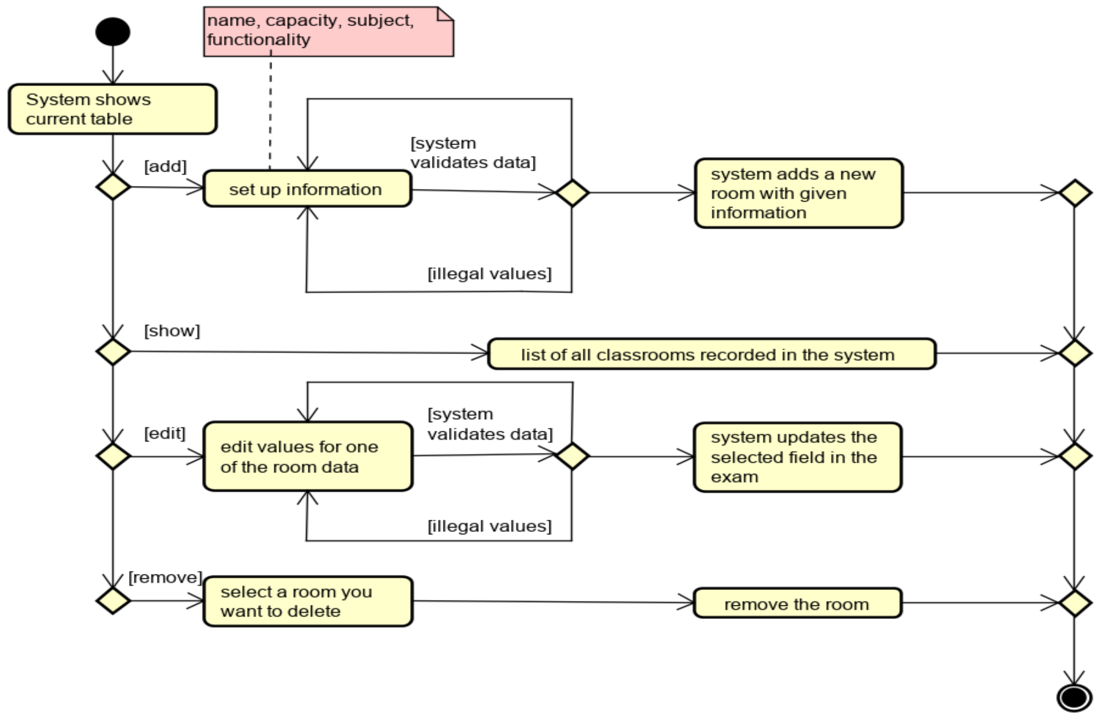

name, capacity, subject, functionality
System shows current table
[add]
set up information
[system validates data]
system adds a new room with given information
[illegal values]
[show]
list of all classrooms recorded in the system
[edit]
edit values for one of the room data
[system validates data]
system updates the selected field in the exam
[illegal values]
[remove]
select a room you want to delete
remove the room

Figure 1 Activity diagram (room)

This is an activity diagram and it represents the switch case for managing the rooms. Other activity diagram can be found in Appendix B.

2.8 Domain Model (Relation)

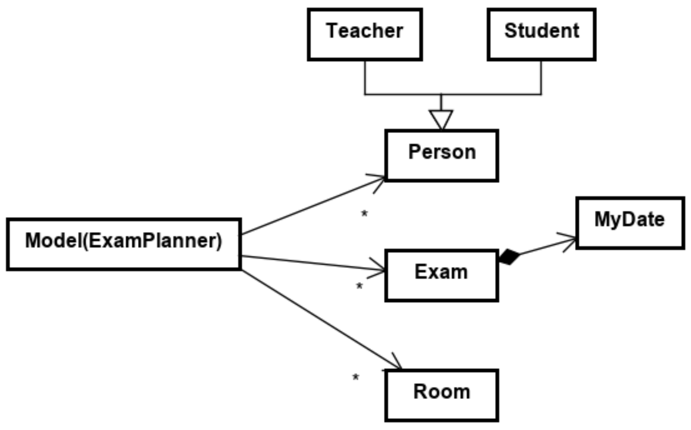


Figure 2 Domain Model (relation)

This diagram is **Domain Model** and it describes basic relations among classes.

3 Design

This section is describing how our system is structured. We implemented architecture, technology and diagrams related to the project.

3.1 Architecture

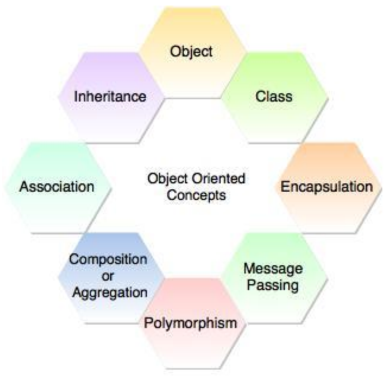


Fig. Object Oriented Concepts

The system follows an Object-Oriented Architecture, this means that the components of the system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

3.2 Technologies

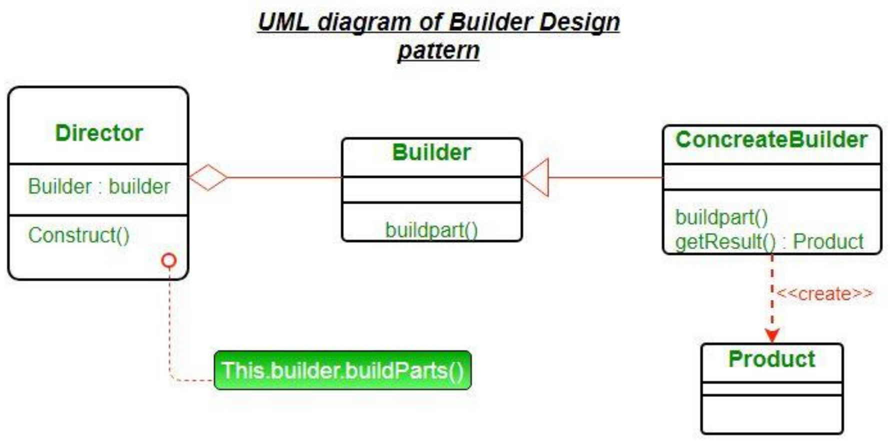
About the technologies used for this project, we have a variety of tools in different categories:

- Web Development:
  - HTML
  - CSS
  - JavaScript
  - Ajax
  - Bootstrap
- Programming Languages:
  - Java

- IDE:
  - IntelliJ IDEA CE
- UML Tool:
  - Astah
- Graphics Tool:
  - JavaFX
- Other:
  - GitHub

3.3 Design Pattern

The design pattern used in this project is called **builder pattern**. As the name implies, it is used to build objects. The builder pattern creates an object step by step, sometimes, the objects created can be complex, made up of several sub-objects or require an elaborate construction process. The exercise of creating complex types can be simplified by using the builder pattern. A *composite* or an *aggregate* object is what a builder generally builds.



3.4 Class Diagrams

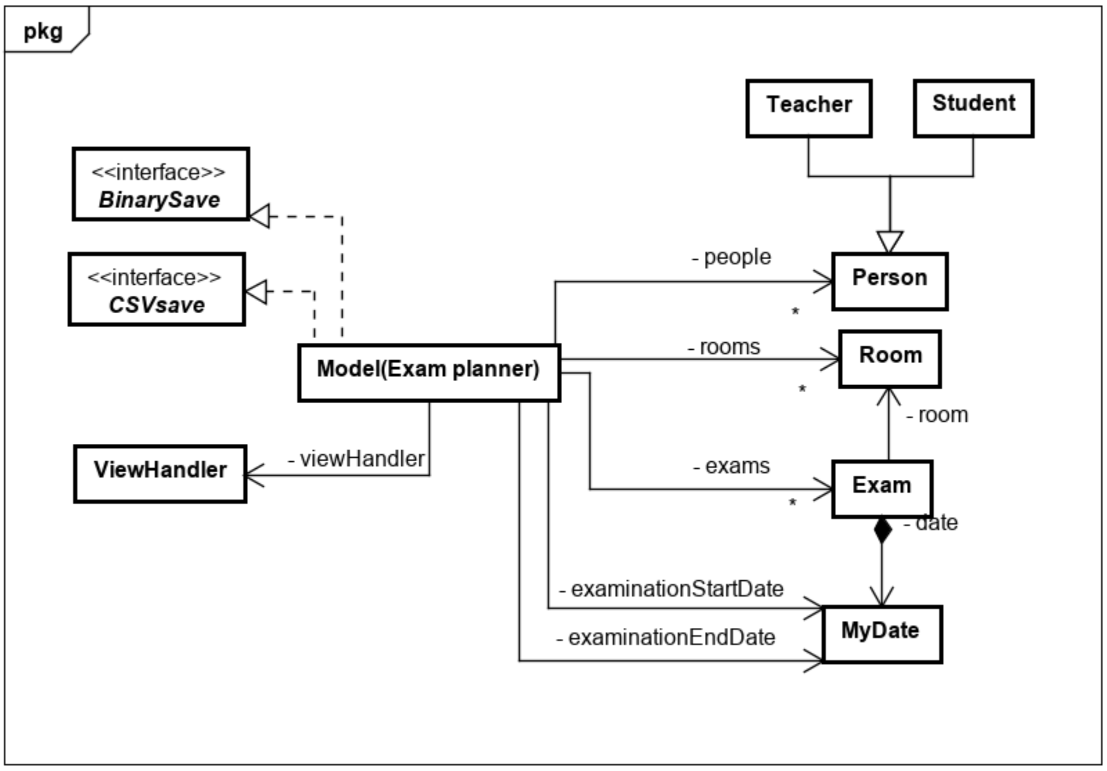


Figure 3 UML (relations)

3. This is a diagram which describes the relations among classes which are explained later. There is one main class called Model (Exam Planner) which has couple associations. Also, it contains inheritances and composition.

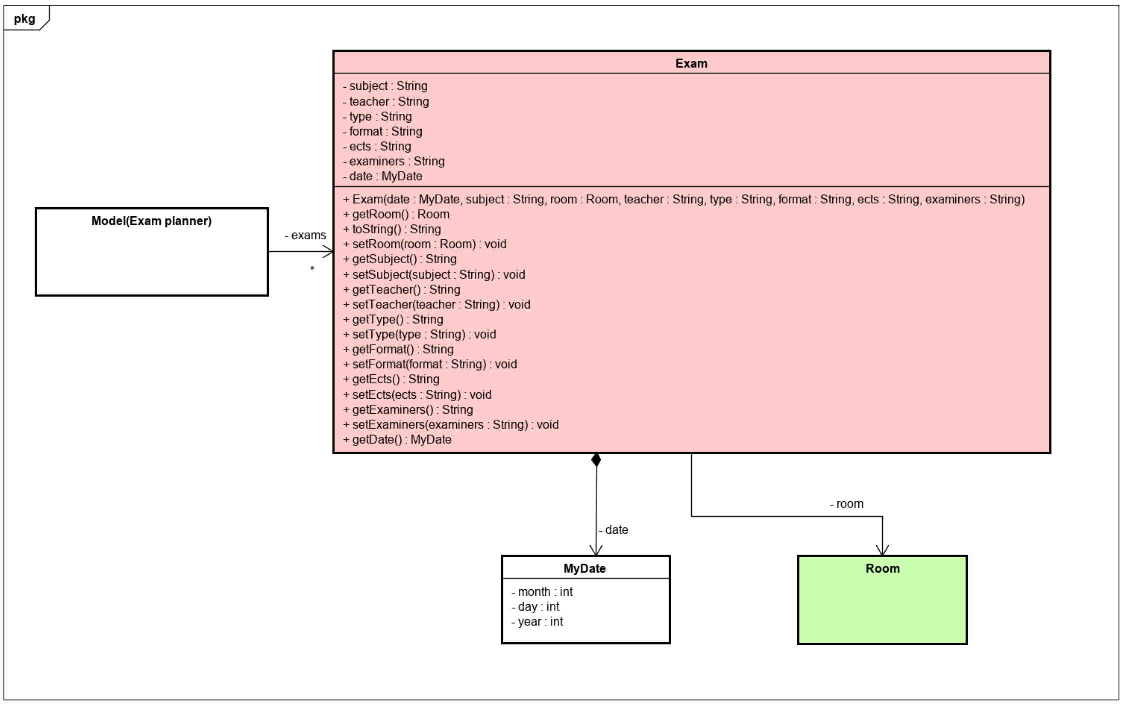

pkg
Exam
- subject : String
- teacher : String
- type : String
- format : String
- ects : String
- examiners : String
- date : MyDate
+ Exam(date : MyDate, subject : String, room : Room, teacher : String, type : String, format : String, ects : String, examiners : String)
+ getRoom() : Room
+ toString() : String
+ setRoom(room : Room) : void
+ getSubject() : String
+ setSubject(subject : String) : void
+ getTeacher() : String
+ setTeacher(teacher : String) : void
+ getType() : String
+ setType(type : String) : void
+ getFormat() : String
+ setFormat(format : String) : void
+ getEcts() : String
+ setEcts(ects : String) : void
+ getExaminers() : String
+ setExaminers(examiners : String) : void
+ getDate() : MyDate
Model(Exam planner)
- exams
*
- date
- room
MyDate
- month : int
- day : int
- year : int
Room

Figure 4 UML (Exams)

4. This UML diagram represents class **Exam** which is a list of exams and it is associated to **Model (Exam Planner).** This **Exam** class has 7 fields of type private. It has a constructor with arguments (*date, subject, room, teacher, type, format, ECTS, examiners).* The methods are only getters and setters. It is using a composition type, where it uses a copy of class **MyDate**. This class is also in association with class **Room**.

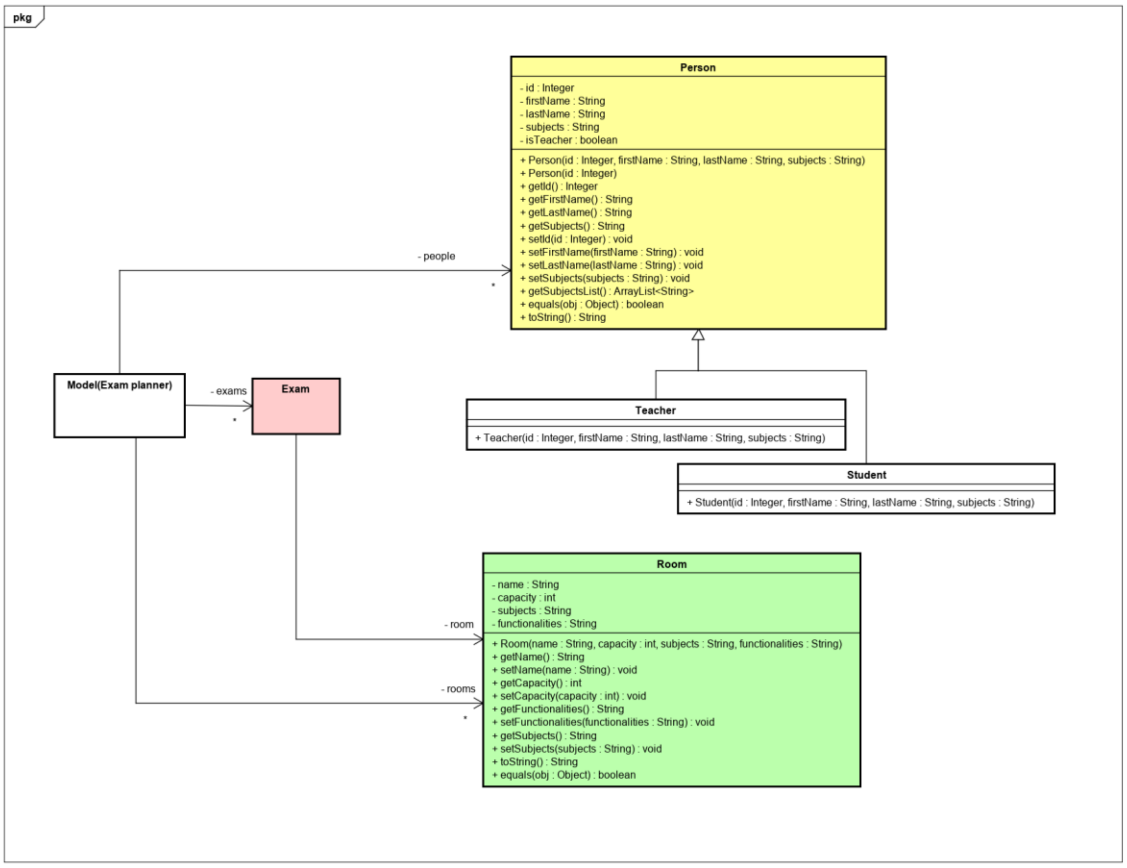

pkg
Person
- id : Integer
- firstName : String
- lastName : String
- subjects : String
- isTeacher : boolean
+ Person(id : Integer, firstName : String, lastName : String, subjects : String)
+ Person(id : Integer)
+ getId() : Integer
+ getFirstName() : String
+ getLastName() : String
+ getSubjects() : String
+ setId(id : Integer) : void
+ setFirstName(firstName : String) : void
+ setLastName(lastName : String) : void
+ setSubjects(subjects : String) : void
+ getSubjectsList() : ArrayList<String>
+ equals(obj : Object) : boolean
+ toString() : String
- people
*
Model(Exam planner)
- exams
*
Exam
Teacher
+ Teacher(id : Integer, firstName : String, lastName : String, subjects : String)
Student
+ Student(id : Integer, firstName : String, lastName : String, subjects : String)
Room
- name : String
- capacity : int
- subjects : String
- functionalities : String
+ Room(name : String, capacity : int, subjects : String, functionalities : String)
+ getName() : String
+ setName(name : String) : void
+ getCapacity() : int
+ setCapacity(capacity : int) : void
+ getFunctionalities() : String
+ setFunctionalities(functionalities : String) : void
+ getSubjects() : String
+ setSubjects(subjects : String) : void
+ toString() : String
+ equals(obj : Object) : boolean
- room
- rooms
*

Figure 5 UML (room, person)

5. This UML represent the associated class **Room,** which was mentioned above. It is associated to class **Model (Exam Planner)** as a list of rooms and to class **Exam**. Class **Room** has 4 fields of type private and one constructor *(name, capacity, subjects, functionalities*).
   Only getters and setters are present here as methods, too. There are two specific methods, method *toString*, and method *equals*, which is of type boolean.
   There is another class **Person** which is a list (people) and it is associated to **Model (Exam Planner)**. **Person** class has 5 fields of type private and 2 constructors (*Id, firstName, lastName, subjects), (Id).* Class **Person** includes **inheritance** of 2 other classes (**Teacher, Student)** with constructors.

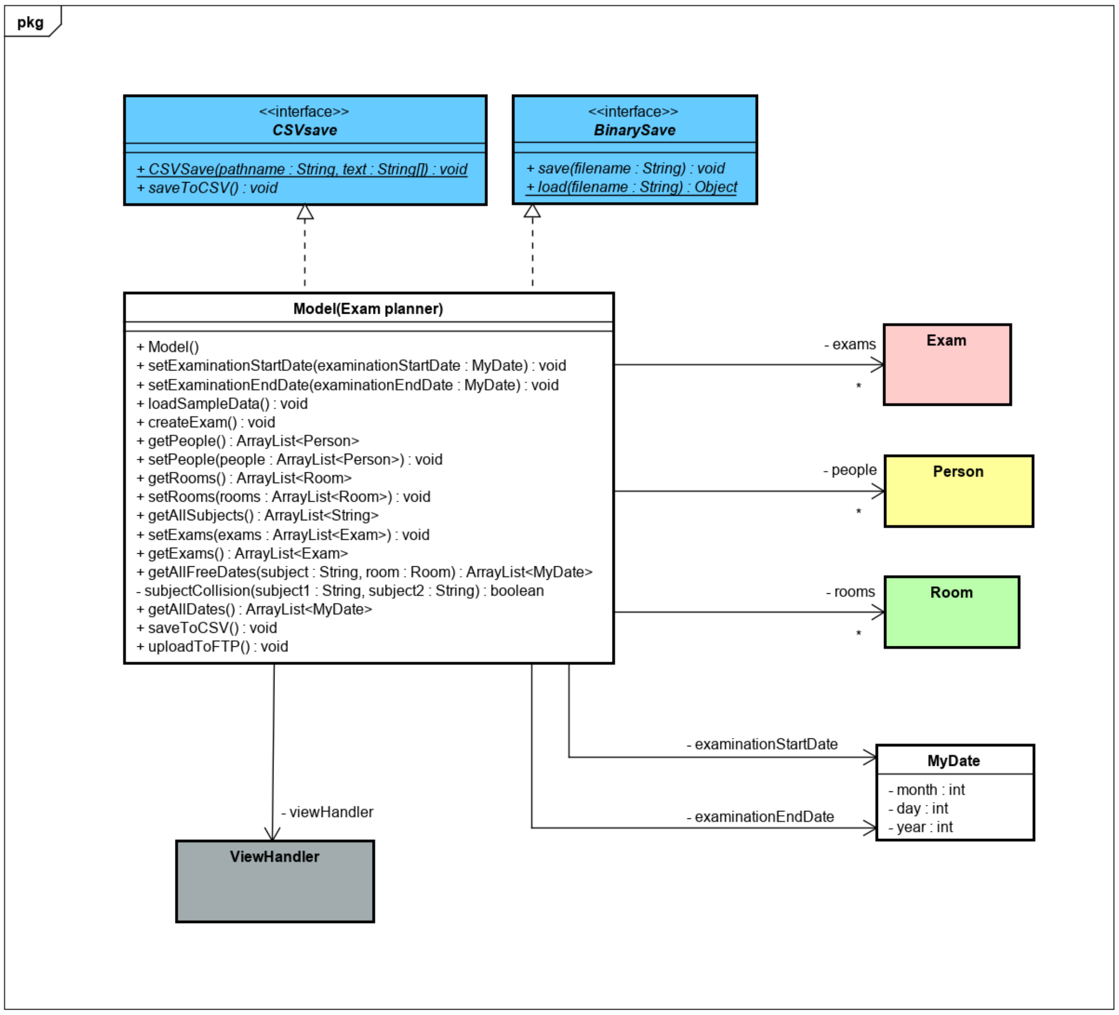

pkg
<<interface>>
CSVsave
+ CSVSave(pathname : String, text : String[]) : void
+ saveToCSV() : void
<<interface>>
BinarySave
+ save(filename : String) : void
+ load(filename : String) : Object
Model(Exam planner)
+ Model()
+ setExaminationStartDate(examinationStartDate : MyDate) : void
+ setExaminationEndDate(examinationEndDate : MyDate) : void
+ loadSampleData() : void
+ createExam() : void
+ getPeople() : ArrayList<Person>
+ setPeople(people : ArrayList<Person>) : void
+ getRooms() : ArrayList<Room>
+ setRooms(rooms : ArrayList<Room>) : void
+ getAllSubjects() : ArrayList<String>
+ setExams(exams : ArrayList<Exam>) : void
+ getExams() : ArrayList<Exam>
+ getAllFreeDates(subject : String, room : Room) : ArrayList<MyDate>
- subjectCollision(subject1 : String, subject2 : String) : boolean
+ getAllDates() : ArrayList<MyDate>
+ saveToCSV() : void
+ uploadToFTP() : void
- exams
*
Exam
- people
*
Person
- rooms
*
Room
- examinationStartDate
- examinationEndDate
MyDate
- month : int
- day : int
- year : int
- viewHandler
ViewHandler

Figure 6 UML (Model)

6. This UML represents class **Model (Exam Planner)** which contains 6 fields of type private, they all are association. 3 of them are of type lists called *exams, people and rooms* (**class Exam, class Person, class Room)** then the **ViewHandler** and class **MyDate**. Also, it contains 2 interfaces, **CSVsave** and **BinarySave**, which save the information.

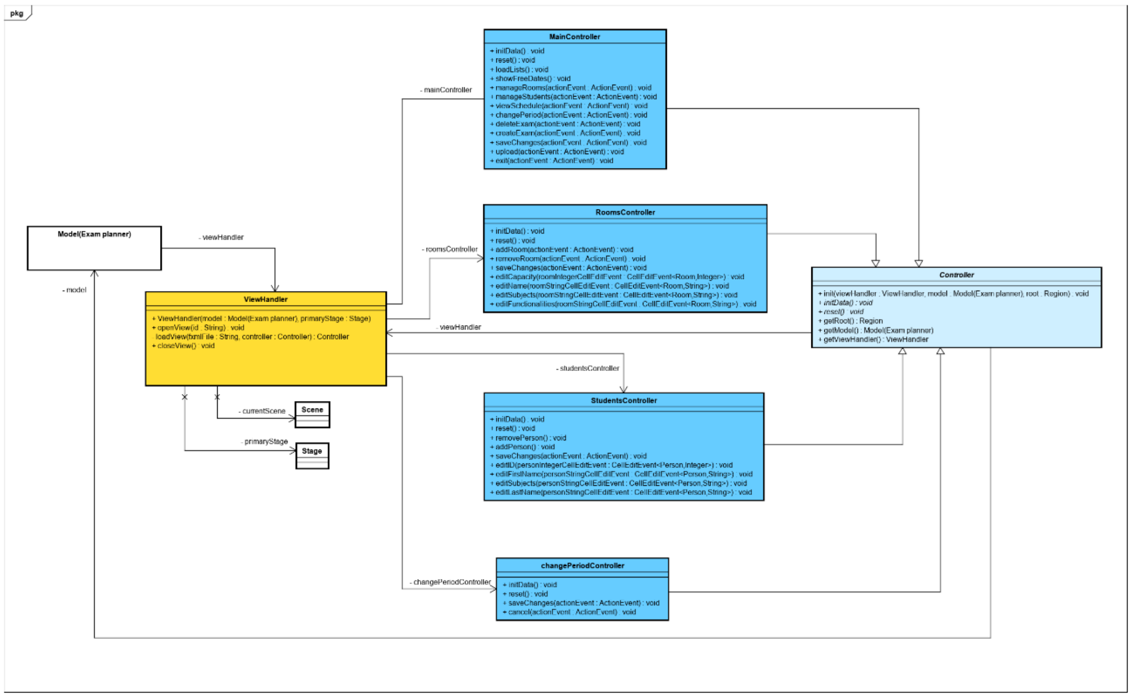
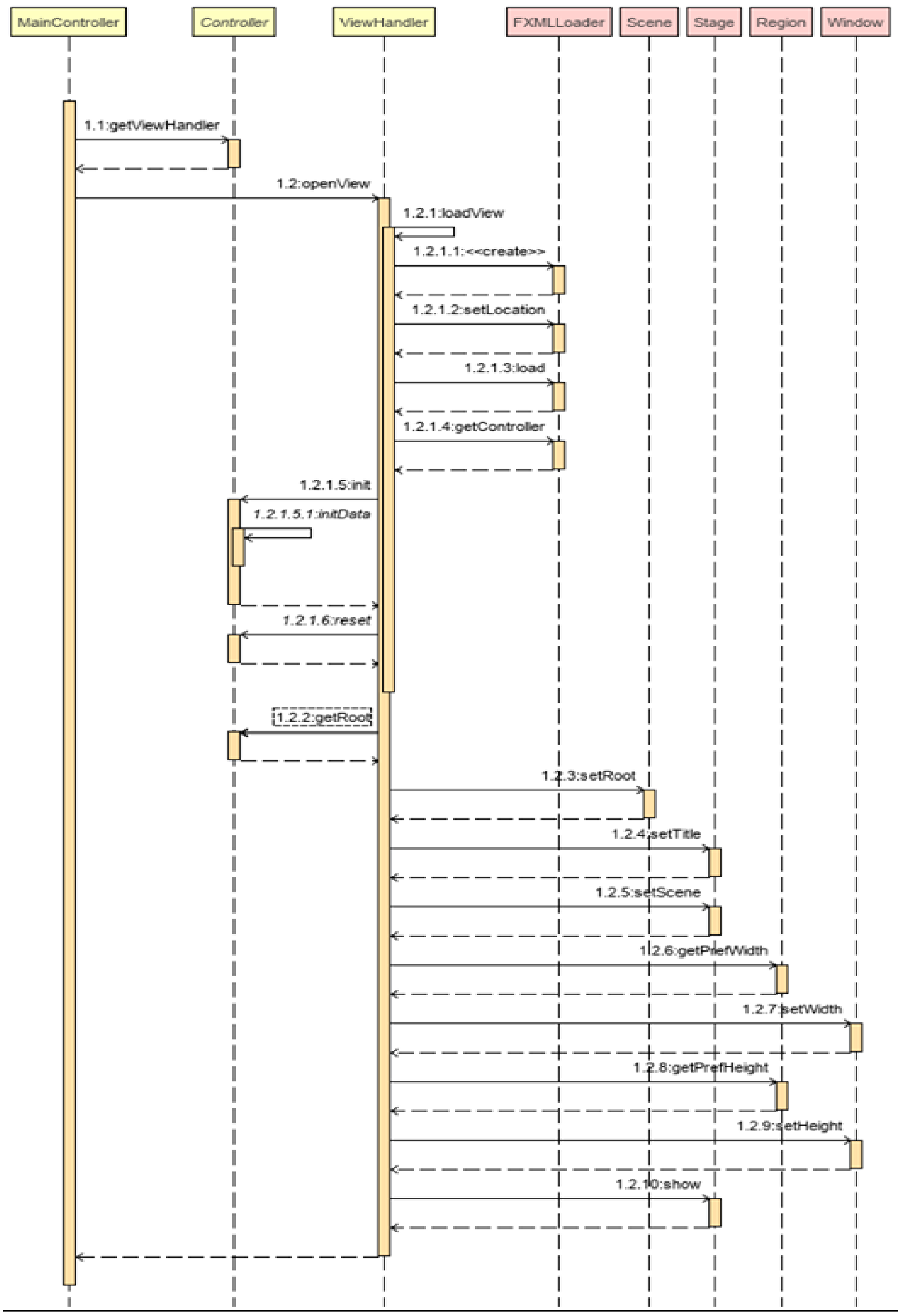

pkg
MainController
+ initData() : void
+ reset() : void
+ loadLists() : void
+ showFreeDates() : void
+ manageRooms(actionEvent : ActionEvent) : void
+ manageStudents(actionEvent : ActionEvent) : void
+ viewSchedule(actionEvent : ActionEvent) : void
+ changePeriod(actionEvent : ActionEvent) : void
+ deleteExam(actionEvent : ActionEvent) : void
+ createExam(actionEvent : ActionEvent) : void
+ saveChanges(actionEvent : ActionEvent) : void
+ upload(actionEvent : ActionEvent) : void
+ exit(actionEvent : ActionEvent) : void
- mainController
RoomsController
+ initData() : void
+ reset() : void
+ addRoom(actionEvent : ActionEvent) : void
+ removeRoom(actionEvent : ActionEvent) : void
+ saveChanges(actionEvent : ActionEvent) : void
+ editCapacity(roomIntegerCellEditEvent : CellEditEvent<Room,Integer>) : void
+ editName(roomStringCellEditEvent : CellEditEvent<Room,String>) : void
+ editSubjects(roomStringCellEditEvent : CellEditEvent<Room,String>) : void
+ editFunctionalities(roomStringCellEditEvent : CellEditEvent<Room,String>) : void
- roomsController
Model(Exam planner)
- viewHandler
- model
ViewHandler
+ ViewHandler(model : Model(Exam planner), primaryStage : Stage)
+ openView(id : String) : void
- loadView(fxmlFile : String, controller : Controller) : Controller
+ closeView() : void
Controller
+ init(viewHandler : ViewHandler, model : Model(Exam planner), root : Region) : void
+ initData() : void
+ reset() : void
+ getRoot() : Region
+ getModel() : Model(Exam planner)
+ getViewHandler() : ViewHandler
- viewHandler
- studentsController
- currentScene
Scene
- primaryStage
Stage
StudentsController
+ initData() : void
+ reset() : void
+ removePerson() : void
+ addPerson() : void
+ saveChanges(actionEvent : ActionEvent) : void
+ editID(personIntegerCellEditEvent : CellEditEvent<Person,Integer>) : void
+ editFirstName(personStringCellEditEvent : CellEditEvent<Person,String>) : void
+ editSubjects(personStringCellEditEvent : CellEditEvent<Person,String>) : void
+ editLastName(personStringCellEditEvent : CellEditEvent<Person,String>) : void
- changePeriodController
changePeriodController
+ initData() : void
+ reset() : void
+ saveChanges(actionEvent : ActionEvent) : void
+ cancel(actionEvent : ActionEvent) : void

Figure 7 UML (GUI)

7. This UML diagram is for *Graphical User Interface* of our project. It has a **ViewHandler** which has several associations and itself it is associated to class **Model (Exam Planner)** and to abstract class **Controller**.

3.5 Interaction diagram

This is a sequence diagram of calling the method "getViewHandler().openView("manageRooms");" in MainController.java



4 Implementation

public ArrayList<MyDate> getAllFreeDates(String subject, Room room)

At first, we create an empty list called "freeDates" where we will later add all the freeDates.
To find a free date for an exam we need to know which subject and what room we want to have the exam in.
A Free Date is such date that:

1. does not use the same room at that specific date
2. There is no exam with same people at that specific date.

For each Date in the examination period the algorithm checks if there is an exam. If there are no exams at that date. The date is "free" and we can add it to our list of free dates. If there is some exam on that date, we need to check if it collides with our exam. It needs to meet our two requirements. The exam cannot use the same room as we want or any of the people that are taking that exam cannot be taking our exam. For the second requirement we can use function subjectCollision(). If both requirements are met, the exam does not collide with the one we want to create so we can add the date to the list of free dates. Otherwise the dates is skipped. When we check all the dates, we end up with a list of all the dates that are possible for the exam we want to create.

private boolean subjectCollision(String subject1, String subject2)

At first, we need two subjects that we want to check. We will call them subject1 and subject2.
Than we can go through every single student and check if the student has both the subjects.
If we find at least one student that has both subject1 and subject2 than those two subjects collide. The function returns the value True. Otherwise if we have not found any such student the return value is False.

public ArrayList<String> getAllSubjects()

This method returns all subjects in an array of Strings. Each student has a string with subjects separated by commas. At first, we create an empty list called "allSubjects". The algorithm goes through all the students and looks at their subjects Strings. Than splits every Subjects String by comma, so that we have Multiple Strings each representing one subject of the student.

We add each of the newly created single subject String to the list "allSubjects, if it is not already in there. We end up with a list of all subjects.

4 Test

The testing strategy that was used was unit testing that focuses on testing each unit implemented in the source code, integration testing focuses on the construction and design of the software and system testing which focuses on the overall performance of the system.

4.1 Test Specifications

Most of the functional and non-functional requirements where implemented on the final version of the program. The team chose not to implement the ones that were not viable and that would not improve the overall function of the software.

5 Results and Discussion

The results were what the team expected for the time that was given which was enough to complete the main objective of the project and add some other features.

6 Conclusions

This system was created to fulfill the requirements given by the customer. The Exam Planner will be helpful, since human cannot avoid making errors while doing complex exam schedules. This project will generate dates, rooms and teacher who are available for specific exam, so that way the software is faster and efficient.

The start of this project was writing an analysis, where every step in process is explained and mentioned requirements are described. In this project, we were able to test almost every requirement. We created diagrams which helped us to imagine the final product and we could easily see the way we wanted to continue.

Followed by design part, we managed to do class diagrams with relations among classes and UML diagrams, which were the base and key to implementation part. By implementing the code, the team came to small errors which were improved.

7 Project future

From a technical viewpoint, the project is a single-user system, which is not a perfect solution, but it is functional for the current client.

As this system will continue to be developed, a client-server system would be a great improvement. This system would be composed of two logical parts: a **server**, which provides services, and a **client**, which requests them. The two parts can run on separate machines on a network, allowing users to access powerful server resources from their personal computers.

Another big improvement would come in the UX/UI department. Since the UI (User Interface) of the system is not deeply worked, using the system feels like using old technology, which worsens the UX (User Experience).

Since this system is made only to fulfill the requirements of VIA University College, a way to scale up the system would be to gather all the fields of information that every university need in order to create an exam schedule and implement them into the project. This way every university could use the same system.

8 Sources of information

Andrew Powell-Morse, 2016, *Waterfall Method, What Is It and When Should I Use It?*...Available at: https://airbrake.io/blog/sdlc/waterfall-model

IEEE Computer Society, 2008. IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation,

Waterfall Methodology in Project Management. Available at: https://www.projectmanager.com/software/use-cases/waterfall-methodology

Architecture available at:
https://www.geeksforgeeks.org/software-engineering-architectural-design/

Technology available at:
https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e